OpenAI 1.0

September 9, 2024

Contents

Future Ideas

fine-turning

OpenAI

The openAI module can be used to understand and generate data specified by the user. For example extract contact information fron an email, correct the grammar from a research paper or get the most important keywords from a long boring text.

:: warning: The AI only understand english words.

## Tokens and price calculation

- Read more about pricing here.
- View your API usage here.

Models

OpenAI comes with a variety of models with different capabilities and price points.

Read more about models here. ### Davinci Davinci is the most caple of all the models. It shines when the task is about performing a lot of understanding of the content, like summarization for a specific audience and creative content generation. It is also very good at solving many kinds of logic problems. However, the Davinci model is the most expensive of all the models.

Good at: Language translation, complex classification, text sentiment, summarization.

Price: $0.0200 / 1k tokens. ### Curie Curies is also very powerful. It can perform many of the same task as Davinci, however with a smaller cost. Curie shines more with nuanced task like sentiment classification and summarization.

Good at: Language translation, complex classification, text sentiment, summarization.

Price: $0.0020 / 1k tokens. ### Babbage Babbage is capable of solving more straight forward task, like semantic Search ranking and how well documents match up with search queries.

Good at: Moderate classification, semanic search classification.

Price: $0.0005 / 1k tokens. ### Ada Ada is the fastes of all the models. Ada is good at performing task that require parsing text, address correction and certain kind of cassification task that don't require too much complexitiy. Ada's capability can often be improved by providing more context.

Good at: Parsing text, simpole classification, address correction, keywords.

Price: $0.0004 / 1k tokens.

Finding the right model

If you don't know what model to use, then you can use this tool, which lets you use different models side-by-side to compare outputs. **Remeber that the tool still uses your token quota.** ## Setting up ### Connect The method `connect` initialises a connection to the openAI API. ### Parameters - `apiKey` is your **secret** API key, which can be found here. - `model` is the model that you want to use. Currently it should be "text-davinci-002", "text-Curie-001", "text-babbage-001" or "text-ada-001".

Example

```
const openAI = Module.load("OpenAI", {version: "1.0.0"})
openAI.connect("YOUR_SECRET_API_KEY", "text-davinci-002")
```

Methods

Grammar correction

The `correctGrammar` method corrects the grammar that the AI receives as input.

Parameters

- `token` the text that the AI should correct.

- `options` an optimal options object. Supported options are;
  - `max_tokens` an integer. The maximum amount of tokens the AI should generate as output. (min: 10, max: 2048, differs from models).
  - `temperature` a double. Determines randomness: Lowering results, the AI will be less creative and the answers will be more repetitive. (min: 0, max: 1).
  - `top_p` a double. Determines diversity via nucleus sampling: 0.5 means halft of all likelihood weighted options are considered. (min: 0, max: 1).
  - `frequency_penalty` a double. How much to penalize new tokens based on their existing frequency in the text so far. Decreses the model's likelihood to repeat the same line verbatim. (min: 0, max 2).
  - `presence penalty` a double. How much to penalize new tokens based on wheater they appear in the text so far. Increases the model's likelihood tio talk about new topics. (min: 0, max: 2).
  - `num_outputs` an integer. Generates multiple outputs. **Use with caution, since it acts as a multiplayer on the number of completions, so this parameter can eat into your token quota very quickly** (min: 1, max: 10). #### Example

```
// load module to disk
const openAI = Module.load("OpenAI", {version: "1.0.0"})
// connect to the open AI API
openAI.connect("YOUR_SECRET_API_KEY", "text-davinci-002", {})
// the specified token which should be corrected
const tokenToBeCorrected = "Dis is vary bad gramma please you help me correct"
// the corrected token
const theCorrectedGrammar = openAI.correctGramma(tokenToBeCorrected)
// completed result
const output = theCorrectedGrammar["result"]
/*
  output: This is very bad grammar. Please help me correct it.
*/
```

Extract TL;DR

The `extractTLDR` method tells the AI to generate a TL;DR (too long; didn't read) string from what the AI receives as input.

Parameters

-`token` the text that the AI will extract TL;DR from. - `options` an optimal options object. Supported options are; - `max_tokens` an integer. The maximum amount of tokens the AI should generate as output. (min: 10, max: 2048, differs from models). - `temperature` a double. Determines randomness: Lowering results, the AI will be less creative and the answers will be more repetitive. (min: 0, max: 1). - `top_p` a double. Determines diversity via nucleus sampling: 0.5 means halft of all likelihood weighted options are considered. (min: 0, max: 1). - `frequency_penalty` a double. How much to penalize new tokens based on their existing frequency in the text so far. Decreses the model's likelihood to repeat the same line verbatim. (min: 0, max 2). - `presence penalty` a double. How much to penalize new tokens based on wheater they appear in the text so far. Increases the model's likelihood tio talk about new topics. (min: 0, max: 2). - `num_outputs` an integer. Generates multiple outputs. **Use with caution, since it acts as a multiplayer on the number of completions, so this parameter can eat into your token quota very quickly** (min: 1, max: 10).

Example

```
// load module to disk
const openAI = Module.load("OpenAI", {version: "1.0.0"})
// connect to the open AI API
openAI.connect("YOUR_SECRET_API_KEY", "text-davinci-002")
// the token that
const javaToken = "One design goal of Java is portability, " +
        "which means that programs written for the Java platform must
            run similarly on any combination of hardware" +
        " and operating system with adequate run time support. " +
        "This is achieved by compiling the Java language code to an
            intermediate representation called Java bytecode, " +
        "instead of directly to architecture-specific machine code.
            Java bytecode instructions are analogous to machine code, "
            +
        "but they are intended to be executed by a virtual machine (VM)
             written specifically for the host hardware. " +
        "End-users commonly use a Java Runtime Environment (JRE)
            installed on their device for standalone Java applications "
             +
        "or a web browser for Java applets."
const tldr = openAI.extractTLDR(javaToken)
const output = tldr["result"]
/*
    output: Java is a portable language that can run on any platform
        with a Java VM.
*/
```

```

// example with temperature set to 1
const tldr = openAI.extractTLDR(javatoken, {temperature: 1})
const output = tldr["result"]
/*
    output: Java code is compiled to an intermediate representation
        called Java bytecode, which is then executed by a virtual
        machine.
    This makes java progams protable, meaning they can run on any
        combination of hardware and operationg system with a Java
        Virtual Machine.
*/
```

Extract keywords

The `extractKeywords` method tells the AI to generate keywords from what the AI receives as input. #### Parameters - `token` the text that the AI will extract keywords from. - `options` an optimal options object. Supported options are; - `max_tokens` an integer. The maximum amount of tokens the AI should generate as output. (min: 10, max: 2048, differs from models). - `temperature` a double. Determines randomness: Lowering results, the AI will be less creative and the answers will be more repetitive. (min: 0, max: 1). - `top_p` a double. Determines diversity via nucleus sampling: 0.5 means halft of all likelihood weighted options are considered. (min: 0, max: 1). - `frequency_penalty` a double. How much to penalize new tokens based on their existing frequency in the text so far. Decreses the model's likelihood to repeat the same line verbatim. (min: 0, max 2). - `presence penalty` a double. How much to penalize new tokens based on wheater they appear in the text so far. Increases the model's likelihood tio talk about new topics. (min: 0, max: 2). - `num_outputs` an integer. Generates multiple outputs. **Use with caution, since it acts as a multiplayer on the number of completions, so this parameter can eat into your token quota very quickly** (min: 1, max: 10). #### Example

```
// load module to disk
const openAI = Module.load("OpenAI", {version: "1.0.0"})
// connect to the open AI API
openAI.connect("YOUR_SECRET_API_KEY", "text-davinci-002", {})
// the token which the AI will extract keywords from
const javaToken = "One design goal of Java is portability, " +
        "which means that programs written for the Java platform must
            run similarly on any combination of hardware" +
        " and operating system with adequate run time support. " +
        "This is achieved by compiling the Java language code to an
            intermediate representation called Java bytecode, " +
        "instead of directly to architecture-specific machine code.
           Java bytecode instructions are analogous to machine code, "
           +
        "but they are intended to be executed by a virtual machine (VM)
            written specifically for the host hardware. " +
        "End-users commonly use a Java Runtime Environment (JRE)
           installed on their device for standalone Java applications "
            +
        "or a web browser for Java applets."

const keywords = openAI.extractKeywords(javaToken)
const output = keywords["result"]
/*
    output: Java, portability, operating system, bytecode, virtual
       machine, JRE
*/
```

Extract contact information

The `extractContactInformation` method generates information that is specified by the user. #### Parameters - `token` the text that the AI will extract contact information from. - `options` an optimal options object. Supported options are; - `max_tokens` an integer. The maximum amount of tokens the AI should generate as output. (min: 10, max: 2048, differs from models). - `temperature` a double. Determines randomness: Lowering results, the AI will be less creative and the answers will be more repetitive. (min: 0, max: 1). - `top_p` a double. Determines diversity via nucleus sampling: 0.5 means halft of all likelihood weighted options are considered. (min: 0, max: 1). - `frequency_penalty` a double. How much to penalize new tokens based on their existing frequency in the text so far. Decreses the model's likelihood to repeat the same line verbatim. (min: 0, max 2). - `presence penalty` a double. How much to penalize new tokens based on wheater they appear in the text so far. Increases the model's likelihood tio talk about new topics. (min: 0, max: 2). - `num_outputs` an integer. Generates multiple outputs. **Use with caution, since it acts as a multiplayer on the number of completions, so this parameter can eat into your token quota very quickly** (min: 1, max: 10). #### Example

```
// load module to disk
const openAI = Module.load("OpenAI", {version: "1.0.0"})
// connect to the open AI API
openAI.connect("YOUR_SECRET_API_KEY", "text-davinci-002", {})
// the token which the AI will look through information
const emailToken = "Good morning Mr.Sheehan, I would like to formally
```

```
    introduce " +
        "myself. My name is Ethan and I am from Secure Shield, a
            company focused on protecting your home " +
        "with security cameras and alarms. We understand the importance
             of keeping your family safe, " +
        "and we want to ensure you have the best security system to
            meet your needs and budget. " +
        "If you are interested in our services, please contact me at
            ccrenshaw@secureshield.com or " +
        "call me at 12-34-56-78. Im looking forward to hearing from you
            !";

// the AI will find the information that corresponds to name, company
    name, services, email and phonenumber
const information = openAI.extractKeywords(javaToken, "name", "company
    name", "services", "email", "phonenumber")
const output = information["result"]
/*
    output: returns a JS Object which contains the information
*/
const name = output["name"] // Ethan
const company = output["company name"] // Secure Shield
const services = output["services"] // Security cameras and alarms
const email = output["email"] // ccrenshaw@secureshield.com
const phonenumber = output["phonenumber"] // 12-34-56-78
```

Get Models

The `getModels` method return a JS object of the available models.

Example

```
// load module to disk
const openAI = Module.load("OpenAI", {version: "1.0.0"})
// gets all models object
const models = openAI.getModels()["models"]

const davinci = models["davinci"]    // text-davinci-002
const curie = models["curie"]        // text-curie-001
const babbage = models["babbage"]    // text-babbage-001
const ada = models["ada"]            // text-ada-001
```

```

// could be used as follows
openAI.connect("YOUR_SECRET_API_KEY", davinci)
```